1 Introduction

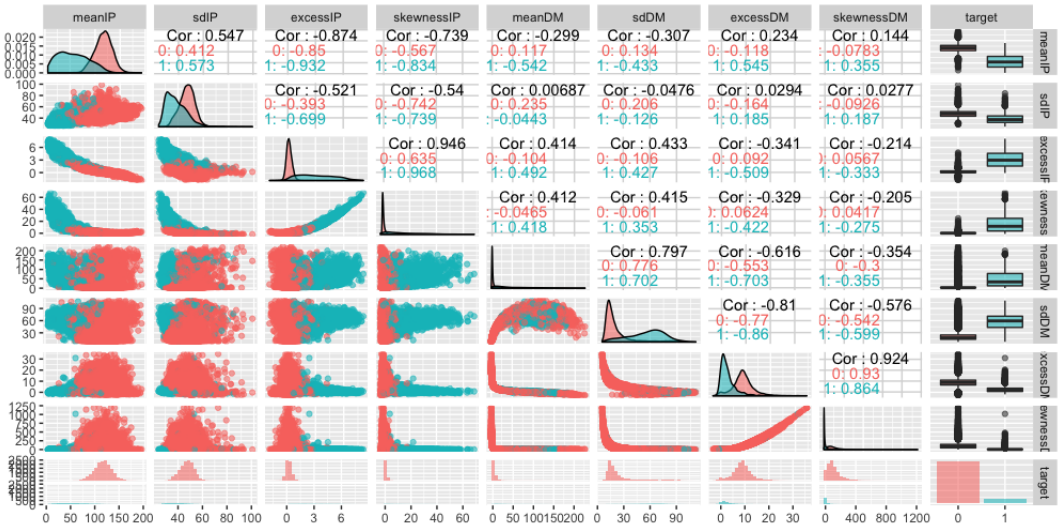
For this report, I am going to preform classification on my data set using more than one method, and then compare which method was the best classifier for my data set. All analysis will be preformed using R software[1].

I am going to be using the pulsar[2] data set. The pulsar data set describes a sample of pulsar candidates collected during the High Time Resolution Universe Survey. Pulsars are a rare type of Neutron star that produce radio emission detectable here on Earth. A potential signal detection is known as a 'candidate'. Each candidate could potentially describe a real pulsar, however in practice almost all detections are caused by radio frequency interference and noise, making legitimate signals hard to find. The data set contains 16,259 spurious examples caused by RFI/noise and 1,639 real pulsar examples. Each candidate is described by 8 continuous variables which are:

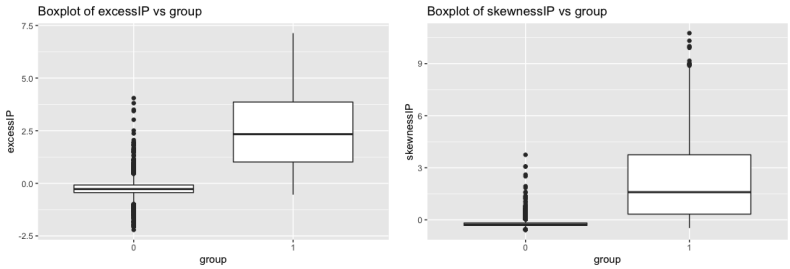
- **meanIP:** Mean of the integrated profile.
- **sdIP:** Standard deviation of the integrated profile.
- **excessIP:** Excess kurtosis of the integrated profile.
- **skewnessIP:** Skewness of the integrated profile.
- **meanDM:** Mean of the DM-SNR curve.
- **sdDM:** Standard deviation of the DM-SNR curve.
- **excessDM:** Excess kurtosis of the DM-SNR curve.
- **skewnessDM:** Skewness of the DM-SNR curve.

I am interested in predicting whether a candidate is really a pulsar, on the basis of the above 8 predictors.

Below is a pairs plot, that I generated using GGally[3] package in R. Pairs plot allows us to see the distribution of single variables and relationships between all pairs of variables. This will help me decide which predictor, or pair of predictors, best separates real pulsars from spurious pulsars.



From the above pairs plot, we see from the box plots that non of the variables totally separates both groups. Although some variables , like excessIP and skewnessIP, are close to doing that. In both boxplots of excessIP and skewnessIP, we see that most of group 0 observations that overlap with group 1 are just the outliers. This is shown more clearly in the boxplots posted below. We can also look for pairs of variables that have high correlation. We see that skewnessIP and excessIP have a high positive correlation. This can also be seen in the pair skewnessDM and ExcessDM . That tells us that candidates that have both high excessIP and skewnessIP are more likely to be real pulsars while candidates with both low excessIP and skewnessIP are more likely to be spurious pulsars.



2 Training/Test Set

Since all the methods that I am going to use are supervised approaches, I am going to split my data into a training and a test set. *Supervised statistical learning* involves building a statistical model for predicting or estimating an output based on one or more inputs[4]. That is, for each observation of the predictor measurements $x_i, i = 1,...,n$ there is an associated response measurement y_i. The *training set* contains labelled observations and is used to build the model that relates the response to the predictors [4]. The observations in the *test set* are unlabelled or treated as such and are used to assess the efficiency of the model.[4]

Since our data set contains a lot of observations, 17,898 observations, a 70:30 train-test split will be efficient. The data set contains 16,259 spurious examples and only 1,639 real Pulsar examples, so I decided to preform stratified random sampling when creating my training set. A *stratified random sample* is one obtained by separating the population elements into non overlapping groups, called *strata*, and then selecting a simple random sample from each stratum.[5]
So for our data set, the two stratas would be the real Pulsar stars and the spurious Pulsar stars. I Randomly selected 70% of the 16,259 spurious stars and 70% of the 1,639 real Pulsar and combined them to form my training set. The remaining 30% observations will form my test set.
To make sure all my work is reproducible, I set set.seed to 1 in R. That is, all my results, graphs, and tables included in the report will be generated with set.seed = 1.

3 The Analysis:

Suppose we observe p-dimensional data vectors $x_1, ..., x_n$ such that $k < n$ of them are from one of G known classes, i.e., $k < n$ are labelled. The estimation of labels for the $n-k$ unlabelled observations is a classification problem.[6]
Through out the report, I am going to generate a classification table for each method and calculate the ARI and missclassification rate. These two numbers will help me decide which method preformed the best. All my classification tables will look like the table below:

	Same group	Different groups
Same group	A	B
Different groups	C	D

Where the columns are the predicted classes, while the rows are the real classes. That is, A observations were correctly predicted to group 1, while C observations were classified into group 1 when they are actually from group 2. So the missclassification rate would just be:

$$MCR: \frac{C+B}{A+B+C+D} \tag{1}$$

ARI is given by

$$ARI = \frac{N(A+D) - [(A+B)(A+C)+(C+D)(B+D)]}{N^2 - [(A+B)(A+C)+(C+D)(B+D)]} \tag{2}$$

where N = A + B + C + D

The lower is the missclassification rate, the better is the method. The smallest missclassification error we can get is 0. The higher is the ARI, the better is the method. The maximum ARI we can get is 1.

3.1 Discriminant Analysis:

Suppose we want to classify an observation into one of K classes. Let π_k represent the prior probability that a randomly chosen observation comes from the kth class. And let $f_k(x) = P(X = x|Y = k)$ denote the density function of X for an observation that comes from the kth class. *Discriminant analysis* uses these information's and Bayes theorem to estimate $P(Y = k|X = x)$.[4]
Bayes theorem states

$$P(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)}$$

We estimate π_k by computing the fraction of the training observations that belong to the kth class. We assume that $f_k(x)$ is drawn from a multivariate normal distribution. That implies that

$$P(Y = k|x = x) = \frac{\pi_k (2\pi)^{-\frac{p}{2}} |\Sigma_k|^{-\frac{p}{2}} exp\{-\frac{1}{2}(x-\mu_k)^T |\Sigma_k|^{-1}(x-\mu_k)\}}{\sum_{l=1}^{K} \pi_l (2\pi)^{-\frac{p}{2}} |\Sigma_l|^{-\frac{p}{2}} exp\{-\frac{1}{2}(x-\mu_l)^T |\Sigma_l|^{-1}(x-\mu_l)\}} \quad (3)$$

3.1.1 LDA:

Linear discriminant Analysis assumes $\Sigma = \Sigma_k$ for $k = 1, ..., K$, then it approximates the Bayes classifier by plugging estimates for π_k, μ_k, and σ^2. These are the the estimates used in LDA:

$$\hat{\mu}_k = \frac{1}{n_k} \sum_{i:y_i=k} x_i; \quad \hat{\Sigma} = \frac{1}{n-K} \sum_{k=1}^{K} \sum_{i:y_i=k} (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T$$

The LDA classifier plugs these estimates into (3) and assigns an observation X = x to the class for which (3) is the largest.
Now I am going to preform LDA on my data set in R using the lda() function from the MASS[7] library. Then I am going to use the predict function on my test set. The predict() function will return a list with three elements. The first element, class, contains LDA's class predictions. This is what I am interested in. So using the predicted classes, I constructed the classification table below:

	0	1
0	4861	17
1	114	378

ARI:	0.81505
missclassification rate	0.02439479

Using the classification table, I calculated the ARI and missclassification rate. From the table above we see that only 17 out of 4878 observation from group 0 were missclassified into group 1, while 114 out of 495 group 1 observations were missclassified into group 0. Here we see a big

problem, we see that a lot of real pulsars (a little above 20%) are being missclassified as spurious.

3.1.2 QDA:

Unlike LDA, *Quadratic discriminant Analysis* assumes each class has its own covariance matrix. That is, it assumes each observation is drawn from a multivariate normal distribution with parameters μ_k and Σ_k.[6] So the QDA classifiers plugs in the estimates for μ_k, Σ_k, and π_k into (3), then assign an observation X = x to the class for which the quantity is the largest. The estimators used in QDA are:

$$\hat{\mu}_k = \frac{1}{n_k} \sum_{i:y_i=k} x_i; \quad \hat{\Sigma} = \frac{1}{n_k} \sum_{i:y_i=k} (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T$$

QDA is implemented in R using the qda() function, which is also part of the MASS library. The predict() function works the exact same way it did for LDA. Using the predicted classes of the test set, I got the classification table below:

	0	1
0	4780	98
1	81	411

ARI:	0.7705925
missclassification rate	0.03333333

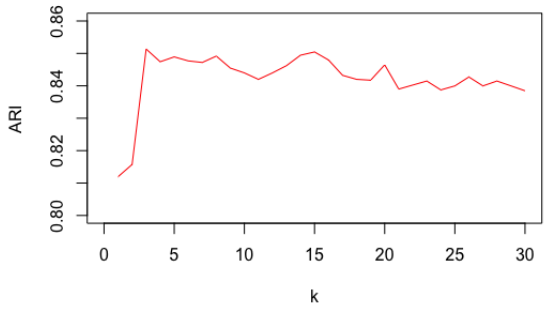
From the ARI and missclassification rate, we see hat QDA preformed worst than LDA. But in this method, less group 1 observations were missclassifed into group 0 (about 16%).

3.2 K-nearest neighbour

In *k-nearest neighbour*, an unlabelled observation is classified based on the labels of the k closest labelled points.[6] The value of k is chosen based on our labelled points. That is, we are going to run kNN algorithm on our training set for more than one k, and choose k that will give us the best classification rate. If different ks give similar classification rates, the smaller value of k is chosen.[6] We will now perform KNN in R using the knn() function, which is part of the class[7] library. The function requires four inputs: a training set, test set, class labels, and a value of k. In R, I ran a for loop for values of k ranging from 1 to 30, and found the ARI of each one of them. Then I chose the k that gave me the highest ARI value. From the plot below, we see that **k = 3** gave the highest ARI value. So using k = 3, I used the predict() function to generate the predicted classes of the test set observations and formed the classification table below:

	0	1
0	4854	24
1	84	408

ARI:	0.8513445
missclassification rate	0.02011173



From the classification table above and the ARI and missclassification error values, we see that the knn method preformed much better than LDA and QDA. Though it did better than LDA, we still see that a lot of real pulsars are being missclassified into spurious pulsars.

3.3 Decision Trees:

In *Decision trees*, we predict that each observation belongs to the most commonly occurring class of training observations in the region to which it belongs.[4] Let node m represents a region R_m with N_m observations, then proportion of observations from class g in node m is $\hat{p}_{mg} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = g)$.[11] All observations in node m are classified into the class with the largest proportion of observations.

Decision trees starts at the top and recursively partitions data based on the best splitter[11]. One way to come up with the splitters is to look at impurity measures. There are three measures that we can look at:

$$\text{classification rate: } E = 1 - max_k(\hat{p}_{mk}) \quad \text{The Gini index: } G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}).$$

$$\text{entropy: } D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$$

When building a classification tree, either the Gini index or the entropy are typically used to evaluate the quality of a particular split, since these two approaches are more sensitive to node purity than is the classification error rate.[4]

In R, I am going to use the function tree() from tree[12] package to generate a decision tree based of the whole data set, the decision tree is shown below. I got the tree to look nice using the package rattle[8] . We see that the function decided that excessIP on its own is enough to split the data. It splitted the data by assigning values less than 0.66 to group 0 and values larger than 0.66 to group 1. We are interested in predicting the performance of this method, so I am going to call the function tree on my train set and predict the classes of my test set observations. The

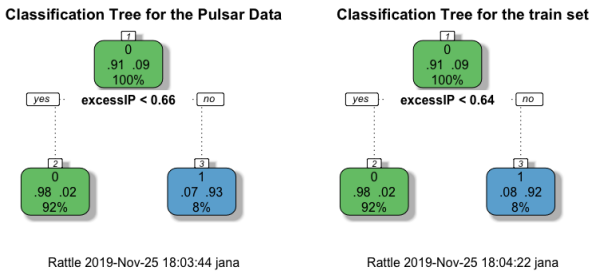


Figure 1: The decision tree for the full data vs the train set

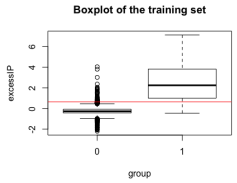


Figure 2: boxplot of excessIP vs group

decision tree we get from applying tree on our train set, shown in figure 1, is really similar to the one generated from the full data set. But instead of excessIP = 0.66, it uses the value 0.64. In figure 2, we see the boxplot of excessIP vs group of our train set. The red line is the line y = 0.64. We see that all the values above the red lines are outliers of group 0. So I assume there wont be a lot of missclassifcations of group 0 observations. But most of the observations of the first quartile of group 1 are being missclassified into group 0, and so I expect that most of our missclassification would be from group 1 observation being missclassified into group 0. The decision tree also let us see that 0.98% of the observations classified into group 0 are actually from group 0 while 2% are observations are from group 1.

I used the predict() function to predict the classes of my test observations and got the classification table shown below:

	0	1
0	4843	35
1	92	400

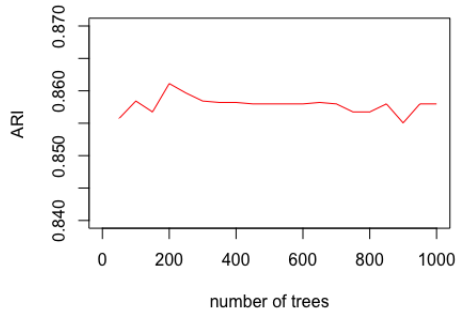
ARI:	0.8262329
missclassification rate	0.02364991

From the above missclassification table, we see that my predictions hold, we see that most of the missclassifications is from group 1 observations being missclassified into group 0.
One of the advantages of decision trees is that they can be displayed graphically, and are easily interpreted. However, decision trees generally do not have the same level of predictive accuracy as some of the other classification approaches. That is because a small change in the data can cause a large change in the final estimated tree.[4]

3.4 Bagging:

Decision trees suffer from high variance. Bagging can be used for reducing the variance of a statistical learning method. Generally, averaging a set of observations reduces variance. So one way to reduce the variance of a statistical learning method is to take many training sets from the population, build a separate prediction model using each training set, and average the resulting predictions.[4] Its not practical for us to generate multiple training sets, so instead we use bootstrapping. *Bootstrap* takes in repeated sample from a single training set. So in bagging, we generate B different bootstrapped training data sets then train our method on the bth bootstrapped training set in order to get $\hat{f}^{*b}(x)$, the predicted class of each of the B trees. We record the class predicted by each of the B trees, and take a majority votes. *Majority vote:* the most commonly occurring class among the B predictions.[4]

We can apply bagging to our data set using the randomForest[9] package in R. The randomForest() function takes in 2 parameters mtry and ntree. mtry is the number of predictors that should be considered for each split of the tree (in bagging we include all predictors), while ntree is the numbers of trees grown. In R, I ran a for loop for several number of trees and recorded the ARI for each different number of trees. I plotted the ARI vs the number of trees and got the plot below. From the plot we see the ntree = 200 gave me the highest ARI. Now I am going to generate my classification table:



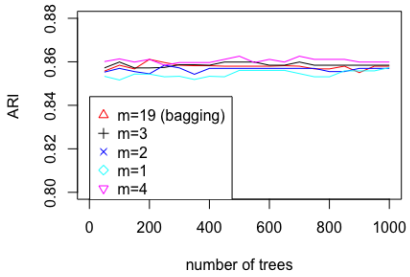
	0	1
0	4856	22
1	80	412

ARI:	0.8596636
missclassification rate	0.01899441

From the above classification table, we se that bagging did better than decision tree, just as expected. But we still see the pattern of a lot of group 1 observations being missclassified into group 0.

3.5 Random Forests:

Random forests provide an improvement over bagged trees by decorrelating the trees.[4] Just like bagging,we build a number of decision trees on bootstrapped training samples. The only difference is that each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors. That is, the split can only use one of those m predictors. A new sample of m predictors is taken at each split. Often, M= $\sqrt{p}$ is used. In R, I am going to apply random forest to my data set using the same function randomForest(). I need to test for different parameters ntree and m. In our data set, we have 8 predictors. $\sqrt{8}$ is approximately 3, but I am also going to test for m=1,2,3, and 4, and see which m gives me the highest ARI out of different number of trees. After running a for loop for the different m's and number of trees, I generated a plot of ARI vs ntree. From the plot below, we see that m = 4 and ntrees = 500 gives the best ARI.



So using these parameters, we can use the predict() function and generate a classification table:

	0	1
0	4856	22
1	78	414

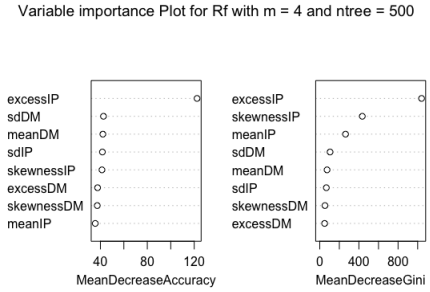
ARI:	0.8625785
missclassification rate	0.01862197

From the ARI and missclassification rate above, we see that random forests so far preformed the best. But again, we still see that pattern of all misclassifications occurring due to group 1 being missclassified into group 0.

3.5.1 Variable Importance

Bagging and Random Forests improved the prediction accuracy of decision trees, but they improved at the expense of interpretability. That is, because in Bagging and Random Forests we are bagging a large number of trees so we wont get an easily interpreted diagram like we did in decision trees. But we can though use the importance() function, and view the importance of each variable. Two measure of variable importance are returned. First is the mean decrease

accuracy and the second is mean decrease Gini: it add up the total amount that the Gini index is decreased by splits over a given predictor, averaged over all B trees.[4] Since Random forest gave me a higher ARI, I am going to use the variable importance plot generated from my above parameters.



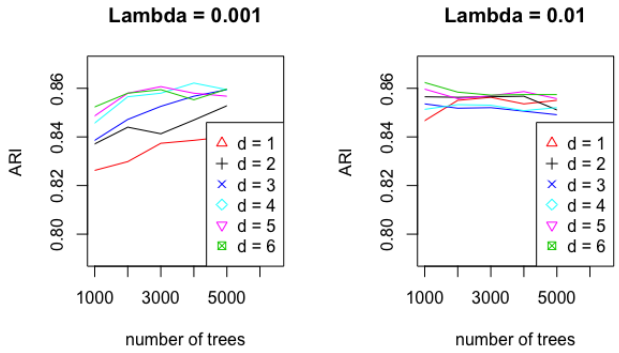
From The plots above, we see that excessIP was the most important variable. This explains why a lot of the missclassifications for both bagging and random forests are due to group 1 being missclassified into group 0. We also see that skewnessDM and excessDM are the least important variables, which was shown in the pairs plot above.

3.6 Boosting:

Boosting is another method that improves the predictions resulting from a decision tree. Boosting works similarly to Bagging, except that the trees are grown sequentially. That is, unlike bagging, boosting does not include bootstrap sampling, but instead each tree is fit on a modified version of the original data set.[4] In R, using gbm[9] package I am going to use the function gbm() to preform boosting on my data set. The function takes on 3 parameters.

1. **number of trees**. Unlike bagging and random forest, a large number of trees can overfit the prediction.
2. **The shrinkage parameter** λ, this controls the rate at which boosting learns. The right choice of λ depends on the problem. It usually take on the values 0.01 and 0.001. Very small λ requires using large number of trees.
3. The number of splits in each tree, **d**. Typically takes on values between 1 and 10.

I ran a for loop for different number of trees and d using $\lambda = 0.01$ and $\lambda = 0.001$. I got the following plots:



From the above plots, we see that $\lambda = 0.01, d = 6$, and ntree $= 1000$ gave me the highest ARI. So now using these parameters, I am going to predict the predicted class of my test observations. The predict function in boosting requires number of trees. So using gbm.perf() function, I got the optimal number of trees for my funcion to be 295. So using all of the above information, I was able to generate the following classification table:

	1	2
0	4858	20
1	79	413

ARI:	0.8636066
missclassification rate	0.01843575

So far, from looking at the ARI and the missclassification rate, we see that boosting preformed the best. But yet again, we see the same pattern of most of the missclassification occurring due to group 1 observations being missclassified into group 0.

3.7 Mixture Discriminant Analysis:

Mixture discriminant analysis preforms model-based clustering on the labelled observations in each known class and then the resulting rule is used to predict the classes for the unlabelled observations.[13] A mixture model is fitted to the labelled observations in each known class and the number of components is determined via some criterion so that $\mathbf{G}_g \geq 1$. A summary of Gaussian Model-Based Discriminant Analysis is found below[13]:

---

for g in 1 to G
  carry out model−based clustering for $x_1,...,x_k$ in class g
  choose a ς_g-component model, e.g., using the BIC
  record corresponding labels LAB_g
end for
compute $\varsigma = \varsigma_1 + \quad + \varsigma_G$
fit a $\varsigma-$component mixture to $x_1,...,x_k$ using labels $LAB_1, \ldots, LAB_\varsigma$
record resulting parameter estimates $\hat{\pi}, \hat{\mu}_1,...,\hat{\mu}_\varsigma, \hat{\Sigma}_1,...,\hat{\Sigma}_\varsigma$
for j = k + 1 to n
  for g in 1 to ς
    compute $\hat{z}_{jg} = \hat{\pi}_g \phi(x_j|\hat{\mu}_g, \hat{\Sigma}_g) / \sum_{h=1}^{\varsigma} \hat{\pi}_h \phi(x_j|\hat{\pi}_h, \hat{\Sigma}_h)$
  end for
  assign x_j to the class corresponding to component $h = argmax_g \hat{z}_{jg}$
end for

In R, I carried out Model-based discriminant analysis using the MclustDA() function found in the mclust[14] package.

Classes	n	%	Model	G
0	11381	90.84	VVV	5
1	1147	9.16	VVV	5

We see from the above table, that for both classes model VVV had the highest BIC and was chosen as the component model. We also see that 5 components were used for each class.

	0	1
0	4791	87
1	72	420

ARI:	0.7951457
missclassification rate	0.029608947

From the ARI and missclassification errors above, we see that this method preformed poorly. Though it had the second lowest ARI from all the methods used, it actually has the lowest missclassifications of group 1 observations. But the percentage of missclassification of group 1 is still much higher than the percentage of missclassification of group 0.

4 Conclusions:

If we look at the results generated above, we see that boosting preformed the best. To reduce the random error, I decided to generate 9 more training sets randomly. So in R, I generated a for loop that sets seed from 2 to 10, and repeated all the methods I did above. Note that I did not preform parameter tuning like what was done above, just reused the same parameters that gave me the best results for set.seed(1). From Table 1 and 2 below, we see that boosting was not always the best method. We see that the best performance was between random forests, and boosting. Wee see that 4 out of 10 times boosting and bagging preformed the best, while 2 out of 10 times bagging preformed the best. From table 3, we can see the average ARI and missclassification rate of the 10 random training set. We see that the average ARI of the random forest was higher than random forests by a really small number. But the average missclassification rate of both methods is the same. So I would conclude that either Random Forests(M= 4, ntree = 500) or boosting(λ = 0.01, d = 6, and ntree = 100) can be used to estimate the class of a Pulsar candidate. As for the worst methods, if we look at table 1 and 2, we see that it was between QDA and mixture discriminant analysis. 6 time out of 10 QDA preformed the worst. But if we look at table 3, the average ARI and missclassification rate, we see that Mixture discriminant had the worst numbers. So I would conclude that we should stay away from these two methods. In all the methods, most of the missclassifications occured in group 1. That is a lot of spurious pulsar candidates are being missclassified as real pulsars.

set.seed	LDA	QDA	Knn	dtree	bagging	random forests	boosting	mixtureDA
1	0.8151	0.7706	0.8513	0.8262	0.8597	0.8626	0.8636	0.7949
2	0.7891	0.7540	0.8330	0.8357	0.8508	0.8534	0.8423	0.7949
3	0.8123	0.7775	0.8423	0.8420	0.8506	0.8496	0.8528	0.7938
4	0.7847	0.7687	0.8155	0.8123	0.8357	0.8306	0.8324	0.7083
5	0.7979	0.7775	0.8326	0.8245	0.8554	0.8503	0.8481	0.7960
6	0.8313	0.7683	0.8539	0.8413	0.8600	0.8608	0.8618	0.7960
7	0.8264	0.7702	0.8497	0.8552	0.8673	0.8649	0.8782	0.7949
8	0.8099	0.7745	0.8355	0.8249	0.8389	0.8452	0.8442	0.7086
9	0.8010	0.7784	0.8401	0.8292	0.8430	0.8522	0.8474	0.7073
10	0.7888	0.7759	0.8445	0.8217	0.8454	0.8474	0.8413	0.7073

Table 1: ARI for different training sets

set.seed	LDA	QDA	Knn	dtree	bagging	random forests	boosting	mixtureDA
1	0.0244	0.0333	0.0201	0.0236	0.0190	0.0186	0.0184	0.0298
2	0.0276	0.0363	0.0227	0.0223	0.0205	0.0201	0.0216	0.0298
3	0.0248	0.0326	0.0214	0.0214	0.0203	0.0203	0.0199	0.0300
4	0.0283	0.0339	0.0253	0.0255	0.0223	0.0231	0.0229	0.0475
5	0.0266	0.0326	0.0229	0.0238	0.0197	0.0205	0.0207	0.0296
6	0.0225	0.0348	0.0201	0.0218	0.0194	0.0192	0.0190	0.0296
7	0.0233	0.0346	0.0209	0.0197	0.0184	0.0188	0.0168	0.0298
8	0.0251	0.0333	0.0225	0.0236	0.0222	0.0212	0.0212	0.0475
9	0.0263	0.0326	0.0220	0.0233	0.0216	0.0203	0.0209	0.0477
10	0.0279	0.0330	0.0216	0.0246	0.0214	0.0210	0.0218	0.0477

Table 2: Missclassification error for different training sets

	LDA	QDA	knn	tree	bagging	rf	boosting	mixtureDA
ARI:	0.8056	0.7716	0.8399	0.8313	0.8507	0.8517	0.8512	0.7602
Missclassification rate:	0.0257	0.0337	0.0220	0.0230	0.0205	0.0203	0.0203	0.0369

Table 3: Average ARI and Missclassification rate of 10 different randomly selected trainings sets

5 References:

1. R Core Team (2019). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria.

2. R. J. Lyon, B. W. Stappers, S. Cooper, J. M. Brooke, J. D. Knowles, Fifty Years of Pulsar Candidate Selection: From simple filters to a new principled real-time classification approach MNRAS, 2016.

3. Barret Schloerke, Jason Crowley, Di Cook, Francois Briatte, Moritz Marbach, Edwin Thoen, Amos Elberg and Joseph Larmarange (2018). GGally: Extension to 'ggplot2'. R package version 1.4.0. https://CRAN.R-project.org/package=GGally

4. James, G., Witten, D., Hastie, T., Tibshirani, R. and Friedman, J. (2013). An Introduction to Statistical Learning. Springer: New York.

5. Scheaffer, Richard L. (2012) Elementary Survey Sampling. Brooks/Cole.

6. McNicholas,S 2019,Introduction to Classification, lecture notes, Data Science STATS 780, McMaster University, delivered 26 Sep. 2019

7. Venables, W. N. Ripley, B. D. (2002) Modern Applied Statistics with S. Fourth Edition. Springer, New York. ISBN 0-387-95457-0

8. Williams, G. J. (2011), Data Mining with Rattle and R: The Art of Excavating Data for Knowledge Discovery, Use R!, Springer.

9. A. Liaw and M. Wiener (2002). Classification and Regression by randomForest. R News 2(3), 18–22.

10. Brandon Greenwell, Bradley Boehmke, Jay Cunningham and GBM Developers (2019). gbm: Generalized Boosted Regression Models. R package version 2.1.5. https://CRAN.R-project.org/package=gbm

11. McNicholas,S 2019,Classification Reggression Trees, lecture notes, Data Science STATS 780, McMaster University, delivered 26 Sep. 2019

12. Brian Ripley (2019). tree: Classification and Regression Trees. R package version 1.0-40. https://CRAN.R-project.org/package=tree

13. McNicholas,S 2019 Classification Using Mixtures, lecture notes, Data Science STATS 780, McMaster University, delivered 8 Nov. 2019

14. Scrucca L., Fop M., Murphy T. B. and Raftery A. E. (2016) mclust 5: clustering, classification and density estimation using Gaussian finite mixture models The R Journal 8/1, pp. 205-233